TDB-ACC-NO: NN950699

DISCLOSURE TITLE: Mechanism for Multiple Source Access of Queues and Stacks

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, June 1995, US

VOLUME NUMBER: 38

ISSUE NUMBER: 6

PAGE NUMBER: 99 - 108

PUBLICATION-DATE: June 1, 1995 (19950601)

CROSS REFERENCE: 0018-8689-38-6-99

DISCLOSURE TEXT:

This document contains drawings, formulas, and/or symbols that will not appear on line. Request hardcopy from ITIRC for complete article.

The objective of this disclosure is to describe a bus attached buffer that performs the functions of multiple FIFO queues and LIFO stacks in a very modular form so that it may be used as one of the elements to create complex, multi-adapter systems to serve as the input-output controller for processor systems or for devices such as DASD, tape, TP, storage systems. The key innovation is the use of the address parameter normally used to specify a memory location as the parameter to specify an individual queue or stack. The real address pointers are managed and checked by the disclosed mechanism. Thus, the interface to the queues and stacks is very simple: the bus attached source/consumer need only read or write to the queue/stack address with a string of data. The disclosed mechanism takes care of the housekeeping and checking for over/under run.

Most DASD, tape, TP and other control units have hardware used in conjunction with a programmable controller to aid in performing the data transfer functions of the control unit. There are many hardware designs and chips that perform FIFO and LIFO functions for data rate matching or buffering. These are limited in size and function and are called two-port buffers, or similar names. Many control units have larger buffers with combined hardware and microcode control that are used primarily in data transfer and are called Automatic Data Transfer, ADT, hardware. In most ADT designs, the hardware generates storage addresses for a Random Access Memory,

RAM, to fetch or store data directly into the RAM. The ADT hardware performs the incrementing of the storage pointers and may check that the pointers do not overrun each other or perform other functions to insure the integrity of the data.

The functions of QUEUES, a FIFO buffer, and STACKS, a LIFO buffer, are illustrated in Fig. 1. In a linear address space such as a RAM, the FIFO is filled by a writer using a WRITE pointer and emptied by a READER using a READ pointer. The WRITE pointer precedes the READ pointer and the data between is considered to be valid. As data is written and read, the two pointers and the space with valid data progresses through the RAM address space. This is illustrated in Fig. 1a. Since RAM is not unlimited, the FIFO uses a limited RAM address. This is shown in Fig. 1b. When either WRITE or READ pointers reach the upper boundary as specified by the TOP pointer, it is set to the lower boundary as specified by the BOTTOM pointer. This is called wrap-around. The data between the READ pointer to the TOP pointer and between the BOTTOM pointer to the WRITE pointer are valid. This is shown in Fig. 1c. A LIFO buffer is shown in Fig. 1d. Data are written at the WRITE pointer and later read at the READ pointer. The TOP and BOTTOM pointers are used to limit the RAM address space used. The LIFO is full when the WRITE pointer reaches the TOP and is empty when the READ pointer reaches the BOTTOM. The algorithms for FIFO and LIFO buffers are well known.

A problem arises when several data sources/consumers use a common bus attached memory as a FIFO or LIFO buffer. This is because each source/consumer has individual ADT hardware that directly addresses the RAM storage. It is difficult to coordinate and check that pointers are not overrunning each other when independent sources and consumers are dynamically using the same portion of RAM. In Fig. 2, users A and B share a portion of linear RAM address space as a FIFO queue buffer. Similarly, C and D share address space for a LIFO stack, and E and F share a FIFO. Because each ADT generates the RAM address for each operation, the users must take care of all of the checking of pointers.

The disclosed mechanism uses the memory address space of bus architecture to perform the ADT functions for multiple sources/consumers. Specific memory addresses serve as the data entry points for programmable FIFO and LIFO buffers. Writers store into the address and readers fetch out of the address. The user transfer

mechanisms are greatly simplified. The linear address space appears to have addresses where data can be stored/fetched in a FIFO or a LIFO. This is illustrated in Fig. 3a. In Fig. 3b, users A, B, C, D, E and F share buffers as in Fig. 2. Each ADT transfers to the buffer address and not to the actual RAM address.

Pointer checking is performed by the disclosed mechanism.

In most bus architectures, a bus user contends for control of the bus and then executes an operation to transfer data. The data transfer is performed by providing an operation code, RAM storage address, and the data. Many bus architectures provide a mechanism for the transfer of multiple units of data called "data streaming" or "burst transfer". This is accomplished by providing the operation code, the initial RAM storage address, and then a stream of data. The RAM storage has a mechanism that increments the RAM address for each unit of data. Each bus user must account for the address of the data to be transferred.

To transfer the next sequential block of data, the bus user must provide a RAM address that is the original RAM address plus the length of the data in the previous transfer to prevent overlaying it. A block format of typical transfers on a bus are illustrated in Fig. 4a. If two or more users are using a segment of RAM as a shared queue or stack, the addresses must be constantly checked to insure that the users do not overlay data on write or fetch invalid data on read. The use of software locks or shared control storage are several mechanisms that have been used to solve this problem. In the disclosed mechanism, these functions are are performed by the unit that controls the RAM storage and thus avoids many of the problems currently encountered by the users of bus attached RAM for shared queues and stacks.

With the disclosed mechanism, users provide the operation code, the address of a FIFO or a LIFO, and the data stream. This is illustrated in Fig. 4b. Note that this is the same format for a regular RAM fetch or store. Subsequent fetch or store to the same buffer uses the same address. The user ADT does not need to manage the RAM addresses. The several users of a FIFO or a LIFO use the same address. From the using units' perspective, they only have to read or write streams of data to the fixed address of the FIFO or LIFO. The disclosed mechanism keeps the specific RAM addresses for each FIFO or LIFO and manages these to read or write the data streams. The mechanism increments/decrements the RAM address for each data unit transferred and checks if readers are overrunning writers, etc.

This structure simplifies the communication needed among the using units and centralizes the increment/decrement and checking hardware. This is illustrated in Fig. 3b where the address space is used for the shared buffers rather that the RAM storage.

Compare this with Fig. 2.

Additional functions to make the disclosed mechanism more usable are disclosed:

1. To provide flexibility of buffer size, function, etc., the buffer functional parameters can be programmed through associated addresses. The specific addresses for each buffer can be programmed.

There can be several addresses associated with each buffer. This permits programming multiple pointers for a given buffer so that multiple readers/writers may share a buffer. An example of use is described later. Buffer TOP and BOTTOM pointer, FIFO or LIFO function are examples of such parameters. Care must be taken when changing these. A lock function is useful for this purpose. TEST and SET is one means of implementing such a lock.

2. In many applications of shared resources such as queues and stacks, many are created but not in active use. Hardware need not be dedicated for each instance but can be time shared. The active registers are kept in a cache in the hardware and the inactive registers are kept in a secure portion of RAM. This is illustrated on Fig. 5.

3. In a FIFO queue, there may be several readers each with a unique pointer. An implementation of this function is illustrated in Fig. 6.

In the illustration, A is writing into a queue while B and C are reading. In general, B and C, are not reading from the same location so B and C must use different queue addresses so the correct pointer is used for their instance of reading. So A writes to address X while B reads from address X1 and C reads from address X2. The mechanism checks that the pointers for X, X1 and X2 do not collide.

4. In the following descriptions all operations are relative to the location of the READER or WRITER pointers. Users cannot directly

manipulate pointers. The disclosed mechanism accounts for wrap around,

5. In a FIFO queue structure it may be desirable to notify create an interrupt when a location relative to the current reader or writer pointer is written or read. This will permit readers or writers to time share other tasks while waiting for a corresponding task to complete its work. The mechanism has a control means so that bus attached users may arm such an interrupt. This is illustrated in Fig. 7a. In a similar manner, a reader or writer can check if there is space for a transfer by querying if the other pointer is a given distance relative from its position, as illustrated in Fig. 7b. In a similar manner, a pointer can be moved a given distance relative to its current position as illustrated in Fig. 7c.

6. In a queue structure used for data transfer or Input/Output operations, it may be desirable to be able to recover or re-read a number of data elements that have been already transferred. This is usually for error recovery purposes. To facilitate this, another storage pointer called "VALID" is defined. This trails the READ pointer and delimits how far "back" the reader may go into the queue and still have valid data. This is illustrated in Fig. 8. The command is SET VALID to READ pointer. This sets the VALID pointer to the current value of the READ pointer. This will permit a read process to protect a portion of the queue from being over written in a circular queue. Another control command is SET READ to VALID pointer. This permits the read process to re-read a segment of the queue in reference to the position of the VALID pointer.

7. In a LIFO buffer, a stack, the reader and writer are pointing to the same element. However, in many cases an entry consists of several elements and care must be taken so that the writer not insert an element of an entry while a reader is taking an element of another entry. A means to serialize the reading/writing of entries is provided. One means is to provide a lock that is set until an entry is completely written or read. Disclosed is a mechanism that creates the effect of ensuring that an entry appears or disappears from the LIFO stack when complete. This permits serialization without locking between reader and writer. This provides a level of robustness against users leaving locks locked. Multiple readers and writers can be separated by giving each a distinct storage address thru which it accesses the stack.

The mechanism is a command to set a pointer to indicate the entry boundary for READ and similarly for WRITE as illustrated in Fig. 9. The entry pointer indicates the beginning of an entry. The reader or writer executes the transfer and then sets the entry boundary to the current reader/writer position. This signals the completion of the entry. If the other process began a transfer, it would begin below the entry pointer as though the other partial entry did not exist. The entry boundary command creates a protected buffer that ensures the atomic addition or removal of an entry even though the actual transfers are not atomic. This mechanism can be implemented for FIFO queues..

8. The disclosed mechanism can be designed to permit sharing the storage across two or more buses. This is illustrated in Fig. 10 where the queue/stack control is across two buses.

This disclosure describes a mechanism that permits loosely communicating, asynchronous data sources and consumers to effectively share a bus connected RAM buffer as multiply addressed FIFO queues and LIFO stacks. The address on the bus is interpreted to address the queue or stack rather than specific storage locations. The logic of the enclosed mechanism keeps the actual RAM addresses, performs the increment/ decrement of address register for data streaming and checks that pointers for readers and writers do not collide. In addition, mechanisms to define the buffer characteristics including the specific addresses for use time share hardware, a set of pointer operations including the definition of a pointer to delimit VALID data and a mechanism to insure atomic insertion/removal of buffer items. Thus, source/consumer adapters cards may be designed for bus attachment and complex systems such as multi-path I/O control units may be developed quickly and with minimum design errors.